IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

In re Patent Application of: Atty Dkt. 550-445
C# M#

EVANS et al TC/A.U.: 2181

Serial No. 10/601,575 Examiner: Vincent Lai

Filed: June 24, 2003 Date: January 17, 2007

Title: SYNCHRONISATION BETWEEN PIPELINES IN A DATA PROCESSING APPARATUS UTILIZING A SYNCHRONISATION QUEUE

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

☐ **Correspondence Address Indication Form Attached.**

☐ **NOTICE OF APPEAL**
Applicant hereby **appeals** to the Board of Patent Appeals and Interferences from the last decision of the Examiner twice/finally rejecting applicant's claim(s). $500.00 (1401)/$250.00 (2401) $

☒ An appeal **BRIEF** is attached in the pending appeal of the above-identified application. **Fee previously paid by check October 30, 2006** $500.00 (1402)/$250.00 (2402) $

☐ Credit for fees paid in prior appeal without decision on merits -$ ( )

☒ Response to Notification of Non-Compliant Appeal Brief. **(no fee)**

☐ Petition is hereby made to extend the current due date so as to cover the filing date of this paper and attachment(s)
One Month Extension $120.00 (1251)/$60.00 (2251)
Two Month Extensions $450.00 (1252)/$225.00 (2252)
Three Month Extensions $1020.00 (1253/$510.00 (2253)
Four Month Extensions $1590.00 (1254/$795.00 (2254) $

☐ "Small entity" statement attached.

Less month extension previously paid on -$( )

TOTAL FEE OF $500.00 PREVIOUSLY PAID BY CHECK ON OCTOBER 30, 2006 0.00

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension. The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this firm) to our **Account No. 14-1140.** A duplicate copy of this sheet is attached.

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:maa

NIXON & VANDERHYE P.C.
By Atty: John R. Lastova, Reg. No. 33,149

Signature: 

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

EVANS et al — Atty. Ref.: 550-445; Confirmation No. 8224

Appl. No. 10/601,575 — TC/A.U. 2181

Filed: June 24, 2003 — Examiner: Vincent Lai

For: SYNCHRONISATION BETWEEN PIPELINES IN A DATA PROCESSING APPARATUS UTILIZING A SYNCHRONISATION QUEUE

* * * * * * * * * * * *

January 17, 2007

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

RESPONSE TO NOTIFICATION OF NON-COMPLIANT APPEAL BRIEF

In response to the Notification dated January 8, 2007, Appellants submit a new Appeal Brief which is believed to respond to the concerns raised in the Notice. Specifically, background material has been removed from the summary and a footnote included encouraging the Board members to review that background material in order to better understand the context and issues related to the claims on appeal. The Examiner has requested additional detail of the queues and tokens. That additional detail has been provided with corresponding references to pages in the specification and figures. To provide much more description of the non-limiting, example, implementations would be to defeat the purpose of the summary. The Examiner and the Board are certainly invited to review the entire specification and figures. The correct Serial No. has been provided on the header of all pages of the Brief.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
John R. Lastova
Reg. No. 33,149

JRL:maa
901 North Glebe Road, 11th Floor
Arlington, VA 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

EVANS et al — Atty. Ref.: 550-445

Serial No. 10/601,575 — Group: 2181

Filed: June 24, 2003 — Examiner: Vincent Lai

For: SYNCHRONISATION BETWEEN PIPELINES IN A DATA PROCESSING APPARATUS UTILIZING A SYNCHRONISATION QUEUE

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT

On Appeal From Final Rejection
From Group Art Unit 2181

John R. Lastova
NIXON & VANDERHYE P.C.
11th Floor, 901 North Glebe Road
Arlington, Virginia 22203-1808
(703) 816-4025
Attorney for Appellant
Evans et al and
ARM Limited

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

EVANS et al — Atty. Ref.: 550-445

Serial No. 10/601,575 — Group: 2181

Filed: June 24, 2003 — Examiner: Vincent Lai

For: SYNCHRONISATION BETWEEN PIPELINES IN A DATA PROCESSING APPARATUS UTILIZING A SYNCHRONISATION QUEUE

* *

January 17, 2007

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a United Kingdom corporation.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals related to this subject application. There are no interferences related to this subject application.

III. STATUS OF CLAIMS

Claims 1-45 are pending and finally rejected.

IV. STATUS OF AMENDMENTS

No amendment has been filed after final. A pre-appeal was filed after final.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The claims recite a data processing apparatus (see e.g., Figure 1) with both a main processor (see e.g., 40 in Figure 1) and a coprocessor (see e.g., 110 in Figure 1) both having plural pipeline stages. Coprocessor instructions in a sequence of instructions to be executed by the data processing apparatus are routed (see e.g., 95 in Figure 1) to the coprocessor for execution. Each coprocessor instruction may be routed through both the main processor pipeline (see e.g., "core" in Figure 3) and the coprocessor pipeline (see e.g., "GCP" in Figure 3) , but this means that the main processor pipeline and the coprocessor pipeline need to be synchronized.

The need for synchronisation stems from the need for interaction between the various pipeline stages of the main processor and the various pipeline stages of the coprocessor during execution of a coprocessor instruction. For example, coprocessor instructions may be cancelled by the main processor if a condition code specified by the coprocessor instruction is not met, or the entire coprocessor pipeline may need to be flushed in the event of a mis-predicted branch that has resulted in the coprocessor

instruction being executed. Further, data may need to be passed between the main processor and the coprocessor in the event that the coprocessor instructions define load or store operations. Page 1, line 27-page 2, line 2.[1]

The claimed synchronization approach uses at least one synchronizing queue to couple a predetermined pipeline stage in one of the pipelines with a partner pipeline stage in the other pipeline. Figure 3 shows examples of multiple synchronization queues used in a non-limiting example implementation described in the specification: an instruction queue 300, a cancel queue 310, a finish queue 320, a length queue 330, an accept queue 340, and a load queue 400. See page 22, line 10-page 24, line 17. Figure 4 shows a more detailed view of the main processor or core and pipeline stages and the queues that connect the two. The core's load/store unit (LSU) 222 is also shown. The LSU accepts store data from the coprocessor via a store queue 400 and generates load data to send to the coprocessor via a load queue 410. Page 24, lines 15-19.

The synchronization queues may, for example, be implemented in the example as FIFO buffers (see Fig. 3 and page 22, lines 11-12), and Figure 6 illustrates details of one example synchronization queue implementation using three registers or buffers 600, 610, and 620. Each register is associated with a flag A, B, and C, respectively, that indicates whether the register contains valid data. New data are moved into the queue by being written into buffer A, i.e. register

[1] The Board is encouraged to consider the helpful background for understanding the context and the problems to which the claims are directed found on pages 1, 2, and 19 of the specification.

600, and continue to move along the queue as long as the next register is empty, or is about to become empty. The multiplexers 660, 670 select the current flag, which then indicates whether the selected output is valid. The state of the buffer flags A, B, and C is used to decide which buffer provides the queue output during each cycle. Page 26, lines 1-25.

The predetermined pipeline stage causes a token to be placed in a synchronizing queue when processing a coprocessor instruction. The partner pipeline stage then processes that coprocessor instruction upon receipt of the token from the synchronizing queue, thereby synchronizing the first and second pipelines at crucial points but also allowing some "slack" between the two pipelines so that strict synchronization at all stages is not necessary. Tokens are generated whenever a coprocessor instruction passes out of a relevant pipeline stage into the next stage. These tokens are picked up by the partner stage in the other pipeline and used to permit the corresponding instruction in that stage to move on. The movement of coprocessor instructions down each pipeline is matched exactly by the movement of tokens along the various queues that connect the pipelines. Page 19, line 27-page 20, line 9.

In a specific non-limiting implementation of the pipeline synchronization technique the main processor or core passes all instructions across to the coprocessor via the instruction queue 300. The coprocessor decodes the instruction and throws it away if it is not a coprocessor instruction (or if it contains the wrong coprocessor number). Each coprocessor instruction then passes down

the pipeline, sending a token down the length queue 330 as it moves into the issue stage. The instruction then remains in the issue stage until it has received a token from the cancel queue 310. If the cancel token does not request that the instruction is cancelled, it moves on to the EX1 stage, placing a token onto the accept queue 340. The instruction then moves down the pipeline until it reaches the EX6 stage. At this point, it waits to receive a token from the finish queue 320, which allows it to retire. See Figure 10 and page 30. Additional features and details are available on pages 30-35 and illustrated in Figures 11-16.

This pipeline synchronization technique can be viewed as a data-driven, loosely-coupled, synchronisation scheme, in contrast to a control-driven, tightly-coupled scheme that involves passing signals with fixed timing between the pipelines. Page 3, lines 15-29.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The rejection on appeal is that of claims 1-45 under 35 U.S.C. §102 as being anticipated based upon U.S. Patent 6,240,508 to Brown.[2]

[2] It is not understood why the Examiner is maintaining the objection to Fig. 2A and to the title. The title was amended as the Examiner requested, and a copy of Fig. 2A as filed was submitted clearly showing that block labeled as MEM1 already has a reference number of 230. Accordingly, these objections should be reversed by the Board.

VII. ARGUMENT

A. The Legal Requirements For Anticipation

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Brown is missing several features from the independent claims.

B. Brown Fails To Teach Multiple Claim Features

1. The Brown Patent

Brown preserves read and write ordering in a macro-pipelined processor of the type that decouples instruction decode and instruction execution so as to allow multiple macroinstructions to exist in the pipeline at various stages of processing. See col. 3, line 50-col. 4, line 14. Figure 1 shows a macro-pipelined processor 10 with an instruction unit 22 (referred to as the I-BOX), an execution unit 23 (referred to as the E-BOX), and a memory management unit 25 (referred to as the M-BOX). See column 7, lines 24-41. In Brown's system, maintaining read and write ordering is essential; otherwise, access to memory is not deterministic. Col. 49, lines 51-57 explain that some memory requests may originate from the E-BOX

23 rather than from the I-BOX 22. As a result, these explicit memory requests must be synchronized with references from previous and subsequent instructions. This is achieved by providing a spec-queue 75 and a spec-queue sync counter within the M-BOX 25. The M-BOX 25 is described in more detail with reference to Figures 15 and 18. See also Brown's claim 1.

2. Brown Lacks A Coprocessor.

The Examiner equates the claimed main processor with Brown's CPU 10 and the claimed coprocessor with Brown's additional CPU 28 shown in Figure 1. But CPU 28 is not described in Brown as a coprocessor for the CPU 10. Col. 35, lines 45-49, referred to by the Examiner, merely describe a memory ownership technique used to ensure one processor can write to a memory location before another processor can read from the same memory location. But a memory ownership technique is not claimed. The Examiner contends that "it is implied by a multiprocessor system that any other processors in the environment can be a coprocessor for the processor in question." Having to rely on "implications" is further evidence that the claimed coprocessor is missing form Brown.

The contention is also incorrect. The normal reason multiple processors are used is to provide additional processing resources in order to improve processing throughput. In such multi-processor systems, each processor works independently of the other rather than as a coprocessor for another processor.

The Examiner also refers to the text at col. 36; line 62-col. 37; line 3 contending that "macro-pipeline considerations mean that instructions are split up amongst different processors." The macro-pipeline is entirely contained in the CPU 10 shown in Figure 1. The referenced text describes the internal operation of the CPU 10, and fails to teach that Brown's additional CPUs 28 act as coprocessors for the CPU 10.

3. Brown Lacks The Claimed Coprocessor.

The claimed coprocessor executes coprocessor instructions appearing in "said sequence of instructions" executed by the main processor. Even if Brown's CPU 28 were erroneously considered to be a coprocessor, Brown's CPU 28 does not execute CPU 28 instructions appearing in *the same sequence of instructions* executed by Brown's CPU 10. The Examiner essentially concedes this point, but references the Examiner's earlier macro-pipelining discussion. How does Brown's macro-pipelining in CPU 10 teach that CPU 28 executes instructions within the same sequence of instructions executed by the CPU 10? Indeed, in a typical multi-processor system, the instructions executed by the CPU 28 would be *different* from the instructions executed by the CPU 10.

4. Each Coprocessor Instruction Is Not Routed Through Both Of The Main and Coprocessor Pipelines.

Claims 1 and 29 recite that each coprocessor instruction is arranged to be routed through *both* the first pipeline (i.e., the pipeline of the main processor) and

the second pipeline (i.e., the pipeline of the coprocessor). Although admitting that Brown's instructions are issued to only one processor, the Examiner nonetheless refers to CPU 10's macro-pipeline. As explained, this macro-pipeline architecture is internal to *a single CPU*—CPU 10. Nowhere does Brown disclose an instruction executed by CPU 10 *also* being executed by CPU 28.

5. Brown Lacks the Claimed Synchronizing Queue.

For the claimed synchronizing queue, the Examiner refers to col.49, lines 58-60, which identifies a spec-queue 75 appearing within an M-BOX 25 of the CPU 10. The spec-queue 75 is provided *entirely within* the main processor 10 to synchronize when the E-BOX 23 can issue memory requests in addition to I-BOX 22. Accordingly, Brown's spec-queue 75 cannot be equated with the claimed "at least one synchronizing queue," because the claimed synchronizing queue "coupl[es] a predetermined pipeline stage in one of the pipelines with a partner pipeline stage in the other of the pipelines," and hence, couples a pipeline stage in the main processor with a pipeline stage in the coprocessor (or vice-versa). No such main processor pipeline to coprocessor pipeline synchronizing queue is described in Brown. Nor does Brown even disclose any queue *between* the CPU 10 and the CPU 28.

6. Brown Lacks the Claimed Token.

The Examiner tries to equate the claimed "token" with the commands associated with Brown's spec-queue 75, which is entirely internal to one CPU 10. Again, the spec-queue 75 does not act as a synchronizing queue between a processor and a coprocessor. The spec-queue commands, generated by the I-BOX and E-BOX of the CPU 10, are used to ensure that memory ordering is preserved when accessing memory. These spec-queue commands are very different from a pipeline synchronizing token placed in a main-coprocessor pipelines synchronizing queue. Nor are Brown's spec-queue commands placed in a synchronizing queue by a predetermined pipeline stage when processing a coprocessor instruction.

Brown also does not teach the claimed partner pipeline stage then processing the coprocessor instruction upon receipt of the claimed token from the synchronizing queue so that the first and second pipelines are synchronized between the predetermined pipeline stage and the partner pipeline stage. This is not surprising since the goal of Brown is to synchronize memory requests. Brown is not focused on synchronizing the pipelines of a main processor and a coprocessor. This entirely different focus in Brown is evident from the following statement found in Brown's Abstract and Summary of the Invention (col. 5, lines 21-25):

> A specifier queue synchronization counter captures synchronization points to coordinate memory request operations among the autonomous instruction decode unit, instruction execution unit, and memory sub-system.

VIII. CONCLUSION

There are multiple claim features missing from Brown. Brown lacks coprocessor instructions from the *same* sequence of instructions as the main processor routed through both the pipelines of the main processor and the coprocessor. Further, Brown lacks a queue by which a synchronizing token is passed between the main processor and a coprocessor. In fact, Brown does not even consider the problem of synchronization between a main processor pipeline and a coprocessor pipeline in situations where a coprocessor instruction is routed through both pipelines. Given the five clear errors in the Examiner's rejection of the independent claims set forth above—any one of which defeats the anticipation rejection based on Brown—the final rejection should be reversed and this application passed to allowance.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: 

John R. Lastova
Reg. No. 33,149

JRL/maa
Appendix A - Claims on Appeal

IX. CLAIMS APPENDIX

1. (original) A data processing apparatus, comprising:

a main processor operable to execute a sequence of instructions, the main processor comprising a first pipeline having a first plurality of pipeline stages;

a coprocessor operable to execute coprocessor instructions in said sequence of instructions, the coprocessor comprising a second pipeline having a second plurality of pipeline stages, and each coprocessor instruction being arranged to be routed through both the first pipeline and the second pipeline; and

at least one synchronising queue coupling a predetermined pipeline stage in one of the pipelines with a partner pipeline stage in the other of the pipelines, the predetermined pipeline stage being operable to cause a token to be placed in the synchronising queue when processing a coprocessor instruction and the partner pipeline stage being operable to process that coprocessor instruction upon receipt of the token from the synchronising queue, thereby synchronising the first and second pipelines between the predetermined pipeline stage and the partner pipeline stage.

2. (original) A data processing apparatus as claimed in Claim 1, further comprising a plurality of said synchronising queues, each said synchronising queue coupling a predetermined pipeline stage in one of the pipelines with a partner pipeline stage in the other of the pipelines.

3. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is an instruction queue, the predetermined pipeline stage is in the first pipeline and is arranged to cause a token identifying a coprocessor instruction to be placed in the instruction queue, and the partner pipeline stage is in the second pipeline and is operable upon receipt of the token to begin processing the coprocessor instruction identified by the token.

4. (original) A data processing apparatus as claimed in Claim 3, wherein the predetermined pipeline stage is a fetch stage in the first pipeline and the partner pipeline stage is a decode stage in the second pipeline, that decode stage being operable to decode the coprocessor instruction upon receipt of the token.

5. (original) A data processing apparatus as claimed in Claim 4, wherein the fetch stage in the first pipeline is operable to cause a token to be placed in the instruction queue for each instruction in the sequence of instructions, and the decode stage in the second pipeline is arranged to decode each instruction upon receipt of the associated token in order to determine whether that instruction is a coprocessor instruction that requires further processing by the coprocessor.

6. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is a cancel queue, the predetermined pipeline stage is in the first pipeline and is arranged to cause to be placed in the cancel queue a token

identifying whether a coprocessor instruction at that predetermined pipeline stage is to be cancelled, and the partner pipeline stage is in the second pipeline and is operable upon receipt of the token from the cancel queue, and if the token identifies that the coprocessor instruction is to be cancelled, to cause that coprocessor instruction to be cancelled.

7. (original) A data processing apparatus as claimed in Claim 6, wherein the predetermined pipeline stage is an issue stage in the first pipeline, and the partner pipeline stage is a stage following an issue stage in the second pipeline.

8. (original) A data processing apparatus as claimed in Claim 6, wherein the partner pipeline stage is operable upon receipt of the token from the cancel queue, and if the token identifies that the coprocessor instruction is to be cancelled, to remove the coprocessor instruction from the second pipeline.

9. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is a finish queue, the predetermined pipeline stage is in the first pipeline and is arranged to cause to be placed in the finish queue a token identifying permission for a coprocessor instruction at that predetermined pipeline stage to be retired from the second pipeline, and the partner pipeline stage is in the second pipeline and is operable upon receipt of the token from the finish queue, and if the token identifies that the coprocessor instruction is permitted to be retired, to cause that coprocessor instruction to be retired.

10. (original) A data processing apparatus as claimed in Claim 9, wherein the predetermined pipeline stage is a write back stage in the first pipeline, and the partner pipeline stage is a write back stage in the second pipeline.

11. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is a length queue, the predetermined pipeline stage is in the second pipeline and is arranged, for a vectored coprocessor instruction, to cause to be placed in the length queue a token identifying length information for the vectored coprocessor instruction, and the partner pipeline stage is in the first pipeline and is operable upon receipt of the token from the length queue to factor the length information into the further processing of the vectored coprocessor instruction within the first pipeline.

12. (original) A data processing apparatus as claimed in Claim 11, wherein the predetermined pipeline stage is a decode stage in the second pipeline, and the partner pipeline stage is a first execute stage in the first pipeline.

13. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is an accept queue, the predetermined pipeline stage is in the second pipeline and is arranged to cause to be placed in the accept queue a token identifying whether a coprocessor instruction in that predetermined pipeline stage is to be

accepted for execution by the coprocessor, and the partner pipeline stage is in the first pipeline and is operable upon receipt of the token from the accept queue, and if the token identifies that the coprocessor instruction is not to be accepted, to cause that coprocessor instruction to be rejected by the main processor.

14. (original) A data processing apparatus as claimed in Claim 13, wherein the predetermined pipeline stage is an issue stage in the second pipeline, and the partner pipeline stage is a second execute stage in the first pipeline.

15. (original) A data processing apparatus as claimed in Claim 14, wherein the partner pipeline stage is operable upon receipt of the token from the accept queue, and if the token identifies that the coprocessor instruction is not to be accepted, to remove the coprocessor instruction from the first pipeline.

16. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is a store queue used when the coprocessor instruction is a store instruction operable to cause data items to be transferred from the coprocessor to memory accessible by the main processor, the predetermined pipeline stage is in the second pipeline and is arranged, when processing one of said store instructions, to cause to be placed in the store queue a token identifying each data item to be transferred, and the partner pipeline stage is in the first pipeline and is operable upon

receipt of each token from the store queue, to cause the corresponding data item to be transferred to the memory.

17. (original) A data processing apparatus as claimed in Claim 16, wherein the predetermined pipeline stage is an issue stage in the second pipeline, and the partner pipeline stage is an address generation stage in the first pipeline.

18. (original) A data processing apparatus as claimed in Claim 1, wherein one of the at least one synchronising queues is a load queue used when the coprocessor instruction is a load instruction operable to cause data items to be transferred from memory accessible by the main processor to the coprocessor, the predetermined pipeline stage is in the first pipeline and is arranged, when processing one of said load instructions, to cause to be placed in the load queue a token identifying each data item to be transferred, and the partner pipeline stage is in the second pipeline and is operable upon receipt of each token from the load queue, to cause the corresponding data item to be transferred to the coprocessor.

19. (original) A data processing apparatus as claimed in Claim 17, wherein the predetermined pipeline stage is a write back stage in the first pipeline, and the partner pipeline stage is a write back stage in the second pipeline.

20. (original) A data processing apparatus as claimed in Claim 18 wherein one of the at least one synchronising queues is a store queue used when the coprocessor instruction is a store instruction operable to cause data items to be transferred from the coprocessor to memory accessible by the main processor, the predetermined pipeline stage is in the second pipeline and is arranged, when processing one of said store instructions, to cause to be placed in the store queue a token identifying each data item to be transferred, and the partner pipeline stage is in the first pipeline and is operable upon receipt of each token from the store queue, to cause the corresponding data item to be transferred to the memory, and wherein the load instruction and store instruction may be vectored coprocessor instructions defining multiple data items to be transferred, and the apparatus further comprises flow control logic, associated with at least one of the load queue and the store queue, operable to send a control signal to the predetermined pipeline stage to stop issuance of tokens by the predetermined pipeline stage whilst it is determined that the associated load or store queue may become full.

21. (original) A data processing apparatus as claimed in Claim 20, wherein the flow control logic is provided for the store queue, the flow control logic being operable to issue the control signal upon receiving an indication from the main processor that the partner pipeline stage cannot accept a data item.

22. (original) A data processing apparatus as claimed in Claim 21, wherein the load queue is a double buffer.

23. (original) A data processing apparatus as claimed in Claim 1, wherein each token includes a tag which identifies the coprocessor instruction to which the token relates.

24. (original) A data processing apparatus as claimed in Claim 23, wherein the main processor is operable, when it is necessary to flush coprocessor instructions from both the first and the second pipeline, to broadcast a flush signal to the coprocessor identifying the tag relating to the oldest instruction that needs to be flushed, the coprocessor being operable to identify that oldest instruction from the tag and to flush from the second pipeline that oldest instruction and any later instructions within the coprocessor.

25. (original) A data processing apparatus as claimed in Claim 24, wherein one or more of said at least one synchronising queues are flushed in response to said flush signal, with the tag being used to identify which tokens within the queue are to be flushed.

26. (original) A data processing apparatus as claimed in Claim 1, wherein the at least one synchronising queue comprises a First-In-First-Out (FIFO) buffer having a predetermined number of entries for storing tokens.

27. (original) A data processing apparatus as claimed in Claim 1, wherein a plurality of said coprocessors are provided, with each synchronising queue coupling a pipeline stage in the main processor with a pipeline stage in one of the coprocessors.

28. (original) A data processing apparatus as claimed in Claim 1, wherein the data processing apparatus has a synchronous design, such that the tokens are caused to be placed in the queue by the predetermined pipeline stage and are caused to be received from the queue by the partner pipeline stage upon changing edges of a clock cycle.

29. (original) A method of synchronisation between pipelines in a data processing apparatus, the data processing apparatus comprising a main processor operable to execute a sequence of instructions and a coprocessor operable to execute coprocessor instructions in said sequence of instructions, the main processor comprising a first pipeline having a first plurality of pipeline stages, and the coprocessor comprising a second pipeline having a second plurality of pipeline stages, and each coprocessor instruction being arranged to be routed through both the first pipeline and the second pipeline, the method comprising the steps of:

(a) coupling a predetermined pipeline stage in one of the pipelines with a partner pipeline stage in the other of the pipelines via a synchronising queue;

(b) placing a token in the synchronising queue when the predetermined pipeline stage is processing a coprocessor instruction;

(c) upon receipt of the token from the synchronising queue by the partner pipeline stage, processing the coprocessor instruction within the partner pipeline stage;

whereby synchronisation of the first and second pipelines between the predetermined pipeline stage and the partner pipeline stage is obtained.

30. (original) A method as claimed in Claim 29, wherein a plurality of said synchronising queues are provided, and said steps (a) to (c) are performed for each synchronising queue.

31. (original) A method as claimed in Claim 29, wherein one of the at least one synchronising queues is an instruction queue, the predetermined pipeline stage is in the first pipeline and the partner pipeline stage is in the second pipeline, the method comprising the steps of:

at said step (b), placing a token in the instruction queue identifying a coprocessor instruction; and

at said step (c), upon receipt of the token, beginning processing of the coprocessor instruction identified by the token within the partner pipeline stage.

32. (original) A method as claimed in Claim 29, wherein one of the at least one synchronising queues is a cancel queue, the predetermined pipeline stage is in the first pipeline and the partner pipeline stage is in the second pipeline, the method comprising the steps of:

at said step (b), placing a token in the cancel queue identifying whether a coprocessor instruction at that predetermined pipeline stage is to be cancelled; and

at said step (c), upon receipt of the token from the cancel queue by the partner pipeline stage, and if the token identifies that the coprocessor instruction is to be cancelled, causing that coprocessor instruction to be cancelled.

33. (original) A method as claimed in Claim 29, wherein one of the at least one synchronising queues is a finish queue, the predetermined pipeline stage is in the first pipeline and the partner pipeline stage is in the second pipeline, the method comprising the steps of:

at said step (b), placing in the finish queue a token identifying permission for a coprocessor instruction at that predetermined pipeline stage to be retired from the second pipeline; and

at said step (c), upon receipt of the token from the finish queue by the partner pipeline stage, and if the token identifies that the coprocessor instruction is permitted to be retired, causing that coprocessor instruction to be retired.

34. (original) A method as claimed in Claim 29, wherein one of the at least one synchronising queues is a length queue, the predetermined pipeline stage is in the second pipeline and the partner pipeline stage is in the first pipeline, and the method comprises the steps of:

at said step (b), for a vectored coprocessor instruction, placing in the length queue a token identifying length information for the vectored coprocessor instruction; and

at said step (c), upon receipt of the token from the length queue by the partner pipeline stage, factoring the length information into the further processing of the vectored coprocessor instruction within the first pipeline.

35. (original) A method as claimed in Claim 29, wherein one of the at least one synchronising queues is an accept queue, the predetermined pipeline stage is in the second pipeline and the partner pipeline stage is in the first pipeline, the method comprising the steps of:

at said step (b), placing in the accept queue a token identifying whether a coprocessor instruction in that predetermined pipeline stage is to be accepted for execution by the coprocessor; and

at said step (c), upon receipt of the token from the accept queue by the partner pipeline stage, and if the token identifies that the coprocessor instruction is not to be accepted, causing that coprocessor instruction to be rejected by the main processor.

36. (original) A method as claimed Claim 29, wherein one of the at least one synchronising queues is a store queue used when the coprocessor instruction is a store instruction operable to cause data items to be transferred from the coprocessor to memory accessible by the main processor, the predetermined pipeline stage is in the second

pipeline and the partner pipeline stage is in the first pipeline, the method comprising the steps of:

at said step (b), when processing one of said store instructions, placing in the store queue a token identifying each data item to be transferred; and

at said step (c), upon receipt of each token from the store queue by the partner pipeline stage, causing the corresponding data item to be transferred to the memory.

37. (previously presented) A method as claimed in claim 29, wherein one of the at least one synchronising queues is a load queue used when the coprocessor instruction is a load instruction operable to cause data items to be transferred from memory accessible by the main processor to the coprocessor, the predetermined pipeline stage is in the first pipeline and the partner pipeline stage is in the second pipeline, the method comprising the steps of:

at said step (b), when processing one of said load instructions, placing in the load queue a token identifying each data item to be transferred; and

at said step (c), upon receipt of each token from the load queue by the partner pipeline stage, causing the corresponding data item to be transferred to the coprocessor.

38. (original) A method as claimed in Claim 37 wherein one of the at least one synchronising queues is a store queue used when the coprocessor instruction is a store instruction operable to cause data items to be transferred from the coprocessor to memory accessible by the main processor, the predetermined pipeline stage is in the second

pipeline and the partner pipeline stage is in the first pipeline, the method comprising the steps of:

at said step (b), when processing one of said store instructions, placing in the store queue a token identifying each data item to be transferred; and

at said step (c), upon receipt of each token from the store queue by the partner pipeline stage, causing the corresponding data item to be transferred to the memory; and

wherein the load instruction and store instruction may be vectored coprocessor instructions defining multiple data items to be transferred, and the method further comprises the step of:

(d) for at least one of the load queue and the store queue, sending a control signal to the predetermined pipeline stage to stop issuance of tokens by the predetermined pipeline stage whilst it is determined that the associated load or store queue may become full.

39. (original) A method as claimed in Claim 38, wherein said step (d) is performed for the store queue, at said step (d) the method comprising the step of issuing the control signal upon receiving an indication from the main processor that the partner pipeline stage cannot accept a data item.

40. (original) A method as claimed in Claim 29, wherein each token includes a tag which identifies the coprocessor instruction to which the token relates.

41. (original) A method as claimed in Claim 40, wherein, when it is necessary to flush coprocessor instructions from both the first and the second pipeline, the method further comprises the steps of:

broadcasting a flush signal from the main processor to the coprocessor identifying the tag relating to the oldest instruction that needs to be flushed;

within the coprocessor, identifying from the tag that oldest instruction and flushing from the second pipeline that oldest instruction and any later instructions within the coprocessor.

42. (original) A method as claimed in Claim 41, further comprising the step of flushing one or more of said at least one synchronising queues in response to said flush signal, with the tag being used to identify which tokens within the queue are to be flushed.

43. (original) A method as claimed in Claim 29, wherein the at least one synchronising queue comprises a First-In-First-Out (FIFO) buffer having a predetermined number of entries for storing tokens.

44. (original) A method as claimed in Claim 29, wherein a plurality of said coprocessors are provided, with each synchronising queue coupling a pipeline stage in the main processor with a pipeline stage in one of the coprocessors.

45. (original) A method as claimed in Claim 29, wherein the data processing apparatus has a synchronous design, such that the tokens are placed in the queue by the predetermined pipeline stage and are received from the queue by the partner pipeline stage upon changing edges of a clock cycle.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.